J Comb Optim (2017) 33:445–472
DOI 10.1007/s10878-015-9973-8

Handling precedence constraints in scheduling problems by the sequence pair representation

Andrzej Kozik[1]

Published online: 6 November 2015

Abstract In this paper, we show that sequence pair (SP) representation, primarily applied to the rectangle packing problems appearing in the VLSI industry, can be a solution representation of precedence constrained scheduling. We present three interpretations of sequence pair, which differ in complexity of schedule evaluation and size of a corresponding solution space. For each interpretation we construct an incremental precedence constrained SP neighborhood evaluation algorithm, computing feasibility of each solution in the insert neighborhood in an amortized constant time per examined solution, and prove the connectivity property of the considered neighborhoods. To compare proposed interpretations of SP, we construct heuristic and metaheuristic algorithms for the multiprocessor job scheduling problem, and verify their efficiency in the numerical experiment.

Keywords Scheduling · Precedence constraints · Sequence pair · Rectangle packing problem

1 Introduction

One of the most frequently appearing constraints in various scheduling fields are precedence constraints saying that a job can be executed if all of its predecessors have already completed their execution. The precedence constraints can model the topology of underlying network of machines, production constraints induced by industrial

✉ Andrzej Kozik
akozik@math.uni.opole.pl

[1] Institute of Mathematics and Informatics, Opole University, pl. Kopernika 11a, 45-040 Opole, Poland

process (Kolisch and Padman 2001) or the orchestration flow in Service Oriented Architecture (SOA)-based systems (Peltz 2003; Świątek et al. 2012).

Given the vast body of applications and interest (eg., Agnetis et al. 2012; Chandra et al. 2014; Debczynski and Gawiejnowicz 2013; Driessel and Monch 2011; Lim et al. 2013; Zhao et al. 2013), in this paper we propose a new method of handling precedence constraints in a general solution-search environment. Contrary to the *a posteriori* technique, in which precedence constraints are verified after solution evaluation, the presented method provides a rigorous structure on the solution neighborhood and efficient algorithms able to filter unfeasible solutions. As a background of the presentation and to illustrate an application example we consider a multiprocessor job scheduling problem under the workspan criterion, but the approach can be easily extended to include release and due dates, malleable and maintenance tasks, as well as other criterion functions (see, Jansen and Zhang 2012; Chen and Chu 2012).

We will show that precedence constraints can be handled using sequence pair (SP) representation of non-overlapping placement of rectangles on a plane (Murata et al. 1996). The SP has many successful industrial applications in the context of physical layout synthesis of VLSI circuits, where it models a placement of transistors, leaf-cells and macro-blocks on a silicon die as a pair of permutations. On the other hand, it was shown in Imahori et al. (2003) that SP can be a solution space for scheduling problems with various definitions of cost functions. Given the importance of precedence constraints, to the best of our knowledge, there was no previous work dealing with feasibility evaluation of arbitrary precedence constraints with SP as a solution space. A restricted form of precedence constraints was considered in the context of rectilinear blocks packing in VLSI (Wakata et al. 2003).

The solution space provided by the SP representation is not free from redundancies, since there is no one-to-one mapping between a schedule and SP. On the other hand, the algorithmic repertoire of computer science includes metaheuristic methods and techniques, in which deficiencies like bigger solution space can be dwarfed by effective exploration of the search space. Examples of such methods are hybrid-metaheuristics (Blum et al. 2008), combining complementary strengths of various techniques to collaboratively tackle hard optimization problems. The recent advent of hyper-heuristic approaches (Burke et al. 2003, 2013), providing a robust upper-level framework to tune and drive underlying heuristics to adapt to features of solved problems, only strengthens this statement.

Nevertheless, complex approaches, able to perform rational and controlled traversal of the solution space, are based on elimination and explorative properties. They include incremental moves and neighborhood structures, which are essential in the context of guiding the search process. Moreover, the efficiency of such methods based on the assumption that one can provide a sequence of moves to every (starting) solution such that every other solution can be reached (especially an optimal one).

The main contribution of this paper is as follows. First, we discuss three interpretations of SP as a scheduling solution, providing three solution-spaces, which differ in size, optimality of representation and complexity of schedule evaluation. For each interpretation, we construct an algorithm that incrementally evaluates an insert neighborhood of SP, computing feasibility of examined solutions in an amortized constant-time per solution. This result is especially important for efficient implemen-

tation of constructive heuristics, Tabu Search, Variable Neighborhood Search as well as other metaheuristics (Blum and Roli 2003), where the time complexity of neighborhood evaluation is critical for the efficiency of the overall methodology, and the naive approach, that *a posteriori* verifies constraint satisfaction for examined solutions is inefficient and unacceptable.

We also prove a connectivity properties of the considered neighborhoods. We show that the results known for the Ulam's metric (Ulam 1972) for permutations[1] can be applied also to the case of precedence constrained permutations—we show an algorithm finding such an optimal transformation. On this basis, we prove connectivity properties for considered precedence-constrained SP neighborhoods. Since presented proofs are constructive, we not only obtain bounds on the diameter of the considered solution spaces, but foremost the resulting algorithms can be independently applied as crossover or path-relinking operators in metaheuristics (Blum and Roli 2003).

To evaluate presented views on SP representation, we constructed heuristics and metaheuristics based on simulated annealing for the considered scheduling problem and, in such a context, verified performance of developed SP neighborhoods in a numerical experiment showing how a choice of SP interpretation impacts the performance of solution search algorithms.

This paper is organized as follows. In Sect. 2, the multiprocessor job scheduling problem is formulated. Section 3 presents details of the SP representation, while an application of SP as a solution space of the scheduling problem and feasibility conditions for the scheduling solution are given in Sect. 4. Next, we construct a incremental precedence constrained SP neighborhood evaluation algorithms, and give neighborhood pruning techniques. Connectivity properties of considered precedence constrained insert neighborhoods of SP are given subsequently. Section 7 is devoted to design and experimental evaluation of solution algorithms for the considered scheduling problem. The last section concludes the paper.

2 Multiprocessor job scheduling problem formulation

There is a set $J = \{1, \ldots, n\}$ of n jobs (tasks) to be scheduled on m identical parallel processors. Each processor can execute at most one job at a time, and each job $j \in J$ requires p_j uninterrupted units of processing simultaneously on q_j processors. Additionally, to maintain a physical proximity of processors allocated to a job, contiguous machine allocations are requested in a schedule. This case is known in the literature as scheduling on a line, adjacent resource system or nonfragmentable multiprocessor system (Turek et al. 1992; Jansen and Thöle 2010; Duin and Sluis 2006) and denoted by $line_j$ in the second field of the three field notation (Graham et al. 1979). Alternatively, q_j can be viewed as a required level of a contiguous resource, see Brucker et al. (1999); Günther et al. (2014); Lodi et al. (2002).

The model assumes that there exist a total ordering of processors (resources), i.e., they can be arranged in a line, and contiguous allocations maximize proximity relation

[1] Ulam's metric, $U(\pi_1, \pi_2)$, is the minimum number of moves required to go from permutation π_1 to permutation π_2, where a move consist of moving a single element of permutation to some other position.

between processors. Possible applications range from assigning memory to processes (Huang and Korf 2013) by the operating system, scheduling of ships on a long wharf (Li et al. 2004), usage of radio frequency spectra (Mitola and Maguire 1999) or scheduling check-in counters at airports (Duin and Sluis 2006).

Additionally, there are given precedence constraints, $\prec$, between jobs. A constraint $i \prec j, i, j \in J$, implies that job i must be completed before job j can be started. Let $G_p(V, E_p)$ be an acyclic directed graph representing precedence constraints between jobs; V is a set of vertices representing jobs and E_p is a set of directed edges such that $(i, j) \in E_p$ if $i \prec j$.

Definition 1 A schedule is feasible if it satisfies all precedence constraints given by G_p.

With respect to a schedule S, the completion time of job i is denoted by C_i, and the makespan is denoted by $C_{max} = \max_{i \in J}\{C_i\}$. The goal is to find a schedule that minimizes the workspan $W_{max} = m \cdot C_{max}$, i.e., a product of the makespan and the number of processors m required in the schedule (Rudek et al. 2013). In the three field notation (Graham et al. 1979), the considered problem can be denoted as $P|prec, line_j|W_{max}$.

The considered workspan objective function reflects the basic scenario of bonding both dimensions of a schedule, yet it has an easy and strong practical explanation, in which a financial cost of renting servers or hiring a crew concerns the whole duration of a schedule (Rudek et al. 2013). The same criterion, a minimal bounding-box of a schedule, has been considered by Korf (2003) as a *total labour cost*. On the one side it does not impose any additional constraints on the schedule and enables to focus on the precedence constraints, on the other hand, much more complex application scenarios are possible. It is easy to construct additional neighborhood filters (Adya and Markov 2003; Augustine et al. 2009; Lodi et al. 2002), based on fixed makespan, number of processors (in the case of discrete vertical dimension) or total resource available (in the case of contiguous job heights), without changing time complexity of presented approach. The same applies if dimensions of a job are to be changed—a fixed set of job configurations can be handled in the constant time. Similarly, a size of malleable task (Jansen and Thöle 2010; Günther et al. 2014) can be chosen during evaluation. The case of maintenance tasks (Wong et al. 2013) can be handled by VLSI approaches to preplaced rectangles (Lai et al. 2001; Young et al. 2004). Computing jobs vertical positions according to Huang and Korf (2013) enables application of the proposed approach to the general multiprocessor scheduling (not constrained to contiguous processors allocations, see e.g. Günther et al. 2014) and resource-constrained project management problems (Brucker et al. 1999; Brucker and Knust 2012). Note, that the considered approach can be also applied for the classical problems, e.g. $Pm|prec|C_{max}$ or $Pm|prec, size_j|C_{max}$, in which each job requires a single processor or $size_j$ of processors and the number of available processors is given or fixed as a part of problem formulation (see, Drozdowski 1996).

The decision version (DP) of the considered workspan minimization problem asks whether a given set of jobs can be scheduled under given precedence constraints with a workspan at most A? To show that DP is NP-complete, we give a reduction from any NP-complete decision problem DP' involving scheduling jobs into

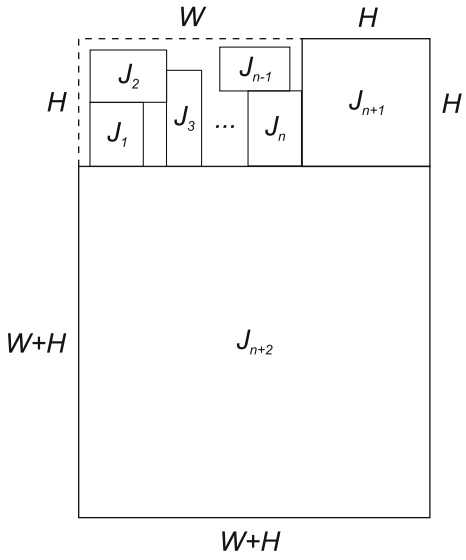

Fig. 1 A schedule corresponding to a certificate of the DP' yes-instance

bounding-box of given width W and height H, e.g., from decision versions of strongly NP-hard $P_{poly}|line_j|C_{max}$ (Jansen and Thöle 2010), in which P_{poly} denotes the number of processors polynomially bounded by the number of jobs, $P|prec|C_{max}$, $P||C_{max}$ (Chen et al. 1999; Svensson 2011) or a variant of strip packing problem (Lodi et al. 2002; Augustine et al. 2009) in the case of contiguous resource instead of discrete processors. Consider an instance of DP': given a set of n jobs with processing times p'_i and required number of processors q'_i (possibly equal to one), $i = 1, \ldots, n$, a precedence constraint graph $G'_p(V, E'_p)$ (possibly empty), does there exist a schedule of length at most W and height (either a maximum resource level or number of processors) at most H? We construct an instance of DP similarly as in Murata et al. (1996). We set the workspan constraint $A = (W + H) \cdot (W + 2H)$ and introduce $n + 2$ jobs: jobs $1, \ldots, n$ from DP', i.e., $p_i = p'_i$ and $q_i = q'_i$, $i = 1, \ldots, n$ and two dummy jobs: job $n + 1$ with $p_{n+1} = q_{n+1} = H$ and job $n + 2$ with $p_{n+2} = q_{n+2} = W + H$. We set the precedence constraints graph G_p with edge set equal to $E'_p \cup \{(i, n + 1)|i = 1, \ldots, n\}$. A schedule of DP corresponding to a certificate for yes-instance of DP' is presented in Fig. 1. It is easy to see that we have a yes-instance to DP' if and only if we have a solution for DP. Otherwise, if the answer for DP is no, then either precedence constraints are not satisfied, or DP' jobs are not fit within intended area.

3 Sequence pair

A SP (Murata et al. 1996) (Γ_+, Γ_-) is a pair of permutations of the set $B = \{1, \ldots, n\}$ of rectangles of width w_i and height h_i, $i \in B$, that encodes pairwise relative non-overlapping placements of rectangles on a plane. The topological relation between each pair of rectangles (i, j), $i, j \in B$, $i \neq j$, imposed by their relative order in the (Γ_+, Γ_-), is either horizontal (1) or vertical (2):

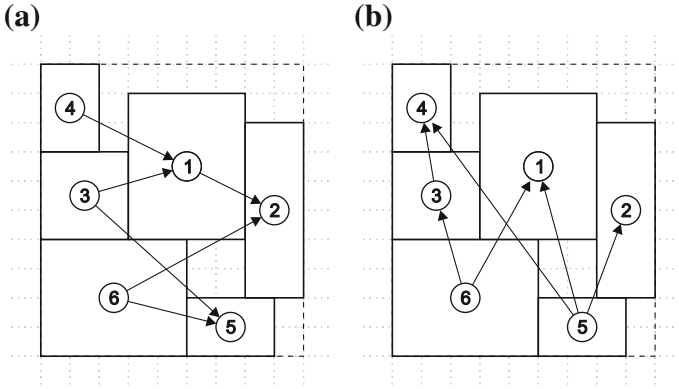

Fig. 2 A packing represented by a SP $(\langle 4, 3, 1, 6, 2, 5\rangle, \langle 6, 3, 5, 4, 1, 2\rangle)$ and its imposed horizontal (**a**) and vertical (**b**) topological relations. Transitive relations were omitted for simplicity

$$(\langle \ldots, i, \ldots, j, \ldots\rangle, \langle \ldots, i, \ldots, j, \ldots\rangle) \Rightarrow i \overset{h}{\rightarrow} j, \tag{1}$$

$$(\langle \ldots, j, \ldots, i, \ldots\rangle, \langle \ldots, i, \ldots, j, \ldots\rangle) \Rightarrow i \overset{v}{\rightarrow} j. \tag{2}$$

Horizontal relation between a pair of rectangles imposes their left-to-right arrangement on a plane, i.e., $i \overset{h}{\rightarrow} j$ forces rectangle j to be placed horizontally on the right of rectangle i. In the case of vertical relations, the arrangement is bottom-up. A placement of rectangles and topological relations imposed by a sample SP are depicted in Fig. 2.

The rectangle packing problem asks for a non-overlapping placement of rectangles on a plane, such that the area of the minimal enclosing rectangle (bounding-box) is minimized. The solution space for the rectangle packing problem provided by SP representation consists of $(n!)^2$ SPs. It is showed in Murata et al. (1996) that each non-overlapping placement of rectangles can be mapped to a SP and each sequence pair can be mapped to a *bottom-left compacted* packing, in which all rectangles are shifted maximally to the left and bottom packing boundaries so that topological relations are met. It is showed, that at least one SP (bottom-left compacted packing) corresponds to an optimal rectangle packing solution.

It is convenient for the analysis to represent relations induced by (Γ_+, Γ_-) in a form of graphs. Let $V = \{1, \ldots, n\}$ denote set of vertices corresponding to rectangles from B and $f : (X, Y) \rightarrow \{(i, j) \in V \times V : i \neq j, (X = \langle \ldots, i, \ldots, j, \ldots\rangle, Y = \langle \ldots, i, \ldots, j, \ldots\rangle)\}$ be a function of SP, returning a set of oriented arcs over V. Consequently, from the SP (Γ_+, Γ_-) we can construct a pair of *constraint graphs*.

Definition 2 Define a *horizontal constraint graph* as $G_h(V, E_h)$, where $E_h = f(\Gamma_+, \Gamma_-)$, i.e., $(i, j) \in E_h$ if $i \overset{h}{\rightarrow} j$.

Definition 3 Define a *vertical constraint graph* as $G_v(V, E_v)$, where $E_v = f(\Gamma_+^R, \Gamma_-)$ and Γ_+^R is the reversed Γ_+ sequence, i.e., $(i, j) \in E_v$ if $i \overset{v}{\rightarrow} j$.

The evaluation of SP, denoted as $\texttt{Evaluate}(\Gamma_+, \Gamma_-)$, is to compute placement coordinates of rectangles. Consider each vertex $j \in V$ has a weight: w_j in G_h and h_j in G_v. Define a length of a path as a sum of weights of vertices on that path. The horizontal (vertical) placement coordinate of top-right corner of rectangle j in a bottom-left compacted packing equals the length of the longest path coming to the vertex j in G_h (G_v), $j \in V$, respectively. Similarly, the horizontal (vertical) span of the packing equals the length of the longest path in G_h (G_v), respectively. Observe that rectangles forming the longest path constraint each other in the same direction and are tightly packed so that any change in their location must produce either overlaps or increase the span of the packing. Therefore, a bottom-left compacted packing is a packing of minimal horizontal and vertical spans that can be obtained from a given SP.

Based on the equivalence between computing longest paths lengths in constraint graphs and longest common subsequences of SP (Takahashi 1996), an optimal, $O(n \log\log n)$ SP evaluation algorithm was presented in Tang et al. (2001) and fully dynamic evaluation algorithms in Kozik (2013). SP has been successfully applied to handle placement constraints in the packing (Lai et al. 2001; Adya and Markov 2003; Young et al. 2004) and rectilinear blocks (Wakata et al. 2003).

4 Sequence pair in precedence constrained scheduling

4.1 Solution space

Consider the SP representation as the solution space for the multiprocessor job scheduling problem, as in Imahori et al. (2003). Each rectangle corresponds to a job, i.e. $B \equiv J$, $w_j = p_j$ and $h_j = q_j$, $j \in \{1, \ldots, n\}$. From a given SP, a schedule is obtained from the evaluated rectangle packing solution. The horizontal span of the packing corresponds to the makespan, C_{max}, and vertical span corresponds to number of processors m. In the further part of the paper, we will use scheduling notion of jobs as rectangles in the packing problem.

First we give a sufficient condition of non-feasibility of a schedule represented by SP.

Property 1 A SP (Γ_+, Γ_-) does not represent a feasible schedule if for some $i \prec j$ an edge (j, i) exist in a horizontal constraint graph G_h imposed by (Γ_+, Γ_-).

Proof Existence of a horizontal relation $j \xrightarrow{h} i$ between jobs, denotes that job i is started after termination of the job j, contradicting a precedence relation $i \prec j$, which cannot be satisfied. □

Definition 4 A horizontal constraint graph G_h imposed by (Γ_+, Γ_-) contradicts a precedence constraint graph G_p, if there exists an edge $(i, j) \in G_h$ s.t. $(j, i) \in G_p$.

Corollary 1 *A SP (Γ_+, Γ_-) can represent a feasible schedule only if G_h imposed by (Γ_+, Γ_-) does not contradicts G_p.*

It is easy to see that Corollary 1 provides only necessary condition, and feasibility of a schedule represented by SP can be answered only after evaluation of a packing solution. Therefore, we propose another interpretations of a SP. Let $G_h^f(B, E_h \cup E_p)$ be a forced horizontal constraint graph, where E_h is an edge set of a horizontal constraint graph G_h imposed by (Γ_+, Γ_-) and E_p is an edge set of precedence constraint graph G_p.

Theorem 1 *A feasible schedule can be obtained from* (Γ_+, Γ_-) *by computing longest paths in* G_v *and* G_h^f*, if* G_h^f *is acyclic.*

Proof For a given SP (Γ_+, Γ_-), its imposed constraint graphs G_v and G_h are evaluated separately to decode a corresponding schedule. The G_v can be evaluated as in the classical rectangle packing. Longest paths in a directed graph can be computed only if it has no cycle (e.g., by an $O(n^2)$ Dijkstra's algorithm). If G_h^f has no cycle, then, after its evaluation, all topological relations imposed by (Γ_+, Γ_-) and all precedence constraints imposed by G_p are satisfied. □

Definition 5 A SP (Γ_+, Γ_-) is called forced-feasible, if it can be evaluated by the algorithm following from Theorem 1.

Using the same argument as in Murata et al. (1996), it can be proved that there exist a forced-feasible SP representing an optimal feasible schedule, which can be evaluated by an algorithm of Theorem 1.

Since the evaluation algorithm following from Theorem 1 requires additional effort, comparing to evaluation of classical bottom-left compacted packings, we relax the assumption that a solution space contains a representation of an optimal schedule. To this end we assume that each SP is evaluated as in the classical rectangle packing.

Definition 6 A SP (Γ_+, Γ_-) is called possibly-feasible, if it conforms to Corollary 1.

Since feasibility of possibly-feasible SP cannot be verified before its evaluation, we further limit the solution space.

Theorem 2 *A SP* (Γ_+, Γ_-) *represents a feasible schedule if the precedence constraint graph* G_p *is a subgraph of horizontal constraint graph* G_h *imposed by* (Γ_+, Γ_-).

Proof According to Definition 1 the graph G_h represents relative horizontal placements between rectangles. In the case of scheduling, existence of a horizontal relation between jobs, $i \xrightarrow{h} j$, denotes that job j is started after termination of the job i, implicitly imposing a precedence relation $i \prec j$. Since G_p is a subgraph of G_h, all precedence constraints defined in G_p are explicitly represented in G_h and satisfied. Therefore, (Γ_+, Γ_-) represents a feasible schedule. □

Definition 7 A SP (Γ_+, Γ_-) is called explicitly-feasible, if it conforms to the assumption of Theorem 2.

Corollary 2 *Let* (Γ_+, Γ_-) *be an explicitly-feasible SP. For each* $(i, j) \in E_P$ $(\Gamma_+, \Gamma_-) = (\langle \ldots, i, \ldots, j, \ldots \rangle, \langle \ldots, i, \ldots, j, \ldots \rangle)$*, i.e. i precedes j in both* Γ_+ *and* Γ_-.

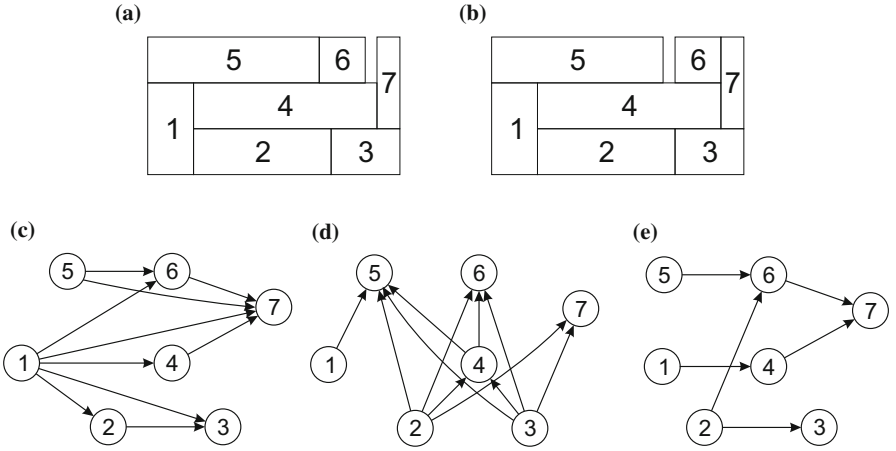

Fig. 3 Interpretation of a SP ($\langle 5, 1, 6, 4, 7, 2, 3\rangle$, $\langle 1, 2, 3, 4, 5, 6, 7\rangle$). *Bottom-left* compacted packing (**a**), forced-feasible schedule (**b**), G_h (**c**), G_v (**d**) and G_p (**e**)

Definition 8 A permutation Γ is called explicitly-feasible if, for each $(i, j) \in E_P$, $\Gamma = \langle \ldots, i, \ldots, j, \ldots \rangle$.

Note that the condition stated in Theorem 2 can give a false-negative result, because even if G_p is not a subgraph of G_h, the precedence constraints can still be satisfied (thought not explicitly), and some solutions cannot be represented in such a solution space (see, Fig. 3b). Apart from the question whether optimal scheduling solution can be always represented by SP and positively verified by Theorem 2, even loosing an optimal scheduling solution from the solution space cannot be considered an issue, since in practical applications often a possibility of finding an optimal solution is sacrificed for the sufficiently-good solution found in a reasonable time (Burke et al. 2003).

Based on the above discussion we can formulate three solution spaces, with:

- forced-feasibility—containing only forced-feasible sequence pairs;
- possible-feasibility—containing only possibly-feasible SPs;
- explicit-feasibility—containing only explicitly-feasible SPs.

In the case of forced- and explicit-feasibility, solution spaces contain feasible solutions only, whereas in the case of possible-feasibility, solution space contain both feasible and unfeasible solutions. Additional differences between these solution spaces are visible in Fig. 3, showing interpretation of SP ($\langle 5, 1, 6, 4, 7, 2, 3\rangle$, $\langle 1, 2, 3, 4, 5, 6, 7\rangle$). Its following constraint graphs are given in Fig. 3c, d, precedence constraint graph is given in Fig. 3e. The SP does not belong to the explicit-feasible solution space, since $2 \prec 6$ is not explicitly satisfied in G_h. The SP belongs to the possibly-feasible solution space, but evaluated as a bottom-left compacted packing (Fig. 3a) represents an unfeasible schedule ($2 \prec 6$ is not satisfied). Finally, the SP is forced-feasible, and can be evaluated to a schedule given in Fig. 3b.

In the further part of the paper, we will prove properties of considered solution spaces and compare their efficiency in a computational experiment.

4.2 Solution perturbation and evaluation of feasibility

Let (Γ_+, Γ_-) be a feasible SP and let $b \in J$ be a job to be rescheduled. Consider an insert move as a solution perturbation, in which a position of job b in Γ_+ and/or Γ_- is changed. Let (Γ'_+, Γ'_-) be a resulting SP and G'_h be its corresponding horizontal constraint graph.

Lemma 1 *A possible-feasibility and explicit-feasibility of solution represented by (Γ'_+, Γ'_-) depends only on edges adjacent to b in a horizontal constraint graph imposed by (Γ'_+, Γ'_-).*

Proof Changing a position of job b in Γ_+ and/or Γ_- can be split into removing of job b from (Γ_+, Γ_-) followed by its re-insertion at the destination positions. Let (Γ^D_+, Γ^D_-) be (Γ_+, Γ_-) with removed job b.

In the graph interpretation removing of job b from (Γ_+, Γ_-) results in removing a vertex b and all its adjacent edges from G_h and G_v constructed from (Γ_+, Γ_-). Let G^D_h be a horizontal constraint graph corresponding to (Γ^D_+, Γ^D_-). According to Definition 1, G^D_h is a subgraph of G_h and is a transitive closure of itself, therefore all precedence constraints from G_p except those adjacent to b are either not contradicted or satisfied explicitly by G^D_h.

Now, let (Γ'_+, Γ'_-) be (Γ^D_+, Γ^D_-) with re-inserted job b. Let G'_h be a horizontal constraint graph corresponding to (Γ'_+, Γ'_-). Since G^D_h is a subgraph of G'_h, therefore G'_h can contradicts or not satisfy constraints from G_p that are adjacent to b, and feasibility depends on edges adjacent to b in G'_h. □

Let E_b be a set of edges adjacent to b in G_p and let $\overline{E_b}$ be E_b with reversed directions of edges. Using Lemma 1 we can state the following lemmas, which enable efficient verification of feasibility of (Γ'_+, Γ'_-).

Lemma 2 *Let (Γ_+, Γ_-) be explicit-feasible, then (Γ'_+, Γ'_-) is explicit-feasible if G'_h contains E_b.*

Lemma 3 *Let (Γ_+, Γ_-) be possibly-feasible, then (Γ'_+, Γ'_-) is possibly-feasible if G'_h avoids $\overline{E_b}$.*

Lemma 4 *Let (Γ_+, Γ_-) be forced-feasible, then (Γ'_+, Γ'_-) is forced-feasible if G'_h avoids E_f, where E_f is a set of edges adjacent to b in a graph $\overline{TC(G(J, E^D_h \cup E_p))}$, E_p is an edge set of G_p and $\overline{TC(G)}$ is a transitive closure of G with reversed direction of edges.*

Proof Let E'_h be an edge set of G'_h. A SP (Γ'_+, Γ'_-) is forced-feasible if $G^*(J, E'_h \cup E_p)$ is acyclic. Since E_p is given and fixed, cyclicality of G^* depends only on G'_h. Since $G^D_h \subset G'_h$ and G^D_h is fixed, we can write $E'_h \cup E_p = E^D_h \cup E^b \cup E_p$, where E^b is a set of edges adjacent to b, resulted from insertion of b into (Γ^D_+, Γ^D_-). From forced-feasibility of (Γ_+, Γ_-) results that $G(J, E_h \cup E_p)$ is acyclic, and since $G^D_h \subset G_h$, $G(J, E^D_h \cup E_p)$ is acyclic as well. Therefore, cyclicality of G^*_h depends only on E^b.

Since G_h^D is transitive closure of itself, therefore, if there exist a path between vertices x and y in G_h^D, an edge $(x, y) \in E_h^D$. Since for any (Γ'_+, Γ'_-), its imposed G'_h is acyclic and $G_h^D \subset G'_h$, E^b cannot contain edges that make a cycle in G_h^*.

Now consider, that G_h^D contains two separated paths (not sharing any vertex) that was connected by an edge from G_p. Let x be the first vertex on that path and let y be the last. Without lost of generality, we may assume that $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, x, \ldots y, \ldots \rangle, \langle \ldots, y, \ldots, x, \ldots \rangle)$. To make a cycle passing through a vertex b, $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, y, \ldots, b, \ldots, x, \ldots \rangle, \langle \ldots, y, \ldots, b, \ldots, x, \ldots \rangle)$, what is not possible for E^b resulting from (Γ'_+, Γ'_-). We have showed, that any two (or more) edges from E^b cannot make $G(J, E_h^D \cup E_p)$ to have a cycle. Therefore, the only possibility is to consider each edge from E^b separately. A single edge (x, b) or (b, y) added to $G(J, E_h^D \cup E_p)$ makes it has a cycle, if there exist a path from x to b or from b to y in $G(J, E_h^D \cup E_p)$, respectively. Consider E_t to be a set of edges adjacent to b in $TC(G(J, E_h^D \cup E_p))$, where $TC(G)$ is a transitive closure of G. Now, for each path beginning or terminating at b in $G(J, E_h^D \cup E_p)$, E_t contains a corresponding transitive edge. For each edge $(x, y) \in E_p$, G_h^* has a cycle only if $(y, x) \in E^b$. Let E_f be E_t with reversed directions. G_h^* has no cycle if it avoids edges from E_f. □

The edge set E_f can be easily computed in $O(n^2)$ by performing forward and backward breadth-first search on $G(J, E_h^D \cup E_p)$ starting from a vertex b.

The insertion of b can be done in $O(n^2)$ possibilities, either by straightforward enumeration and evaluation of all insertions or by incremental evaluation of SP insert neighborhood algorithm presented in the next section.

5 The Precel algorithm

In this section, we develop the Precel algorithm, which evaluates feasibility of all insert moves of a job in a pair of sequences. According to Lemmas 2–4, verifying feasibility of a SP solution with considered interpretations involves checking if some given graph is either contained or avoided in the horizontal constraint graph imposed by SP. Therefore, the Precel algorithm has two variants. The first variant evaluates a SP as feasible, if its imposed horizontal constraint graph contains a given edge set E. We call such SP a *c-feasible*. The second variant evaluates a SP as feasible, if its imposed horizontal constraint graph avoids a given edge set E. We call such SP a *a-feasible*. We present the algorithm in the form of Janiak et al. (2010) to ease integration in the case of explicit-feasible solution space. Presented results can also be combined with neighborhood evaluation techniques described in Imahori et al. (2005). In the rest of the section we show properties of SP that enable efficient verification of c-feasibility (Lemma 5) and a-feasibility (Lemma 6). Then, we give an outline of neighborhood evaluation schedule, which enables efficient amortization of feasibility evaluation complexity. The resulting Precel algorithm and neighborhood pruning properties are presented next.

5.1 Feasibility evaluation of SP

Let (Γ_+, Γ_-) be a SP representing schedule of jobs from J and let $b \in J$ be an arbitrarily chosen job.

Definition 9 (*b-split*) Define b-split of (Γ_+, Γ_-) as $\langle \beta_h, \widetilde{\beta}_h, \beta_v, \widetilde{\beta}_v \rangle$, where $\beta = J \backslash \{b\}$ and

$$\beta_h = \{i \in \beta | i \overset{h}{\rightarrow} b\},\ \widetilde{\beta}_h = \{i \in \beta | b \overset{h}{\rightarrow} i\},$$
$$\beta_v = \{i \in \beta | i \overset{v}{\rightarrow} b\},\ \widetilde{\beta}_v = \{i \in \beta | b \overset{v}{\rightarrow} i\}.$$

Informally, the b-split divides jobs surrounding b (in the schedule viewed as a placement of rectangles on a plane) into four subsets: placed on the left to job b (in β_h), placed on the right ($\widetilde{\beta}_h$), below (β_v) and above ($\widetilde{\beta}_v$). Let $\mathcal{N}^- = \{i \in J : (i, b) \in E\}$ and $\mathcal{N}^+ = \{i \in J : (b, i) \in E\}$.

Lemma 5 *Let (Γ_+, Γ_-) be a c-feasible SP and let (Γ'_+, Γ'_-) be a modification of (Γ_+, Γ_-), in which a position of job b in Γ_+ and Γ_- is changed. Let $\langle \beta_h, \widetilde{\beta}_h, \beta_v, \widetilde{\beta}_v \rangle$ be b-split of (Γ'_+, Γ'_-). Let Λ_{β_h} be a number of not contained edges $(i, b) \in E$, i.e. $\Lambda_{\beta_h} = \left|\mathcal{N}^- \setminus \beta_h\right|$, and let $\Lambda_{\widetilde{\beta}_h}$ be a number of not contained edges $(b, i) \in E$, i.e $\Lambda_{\widetilde{\beta}_h} = \left|\mathcal{N}^+ \setminus \widetilde{\beta}_h\right|$. The (Γ'_+, Γ'_-) is c-feasible if $\Lambda_{\beta_h} = 0$ and $\Lambda_{\widetilde{\beta}_h} = 0$.*

Proof Since (Γ_+, Γ_-) is a c-feasible SP, according to Lemma 1, in (Γ'_+, Γ'_-) all edges from E except (possibly) those adjacent to b are contained. If $\Lambda_{\beta_h} = 0$ and $\Lambda_{\widetilde{\beta}_h} = 0$ then all edges from E adjacent to b are contained as well and (Γ'_+, Γ'_-) is c-feasible. □

Similarly we can prove the following lemma for the a-feasible case.

Lemma 6 *Let (Γ_+, Γ_-) be a a-feasible SP and let (Γ'_+, Γ'_-) be a modification of (Γ_+, Γ_-), in which a position of job b in Γ_+ and Γ_- is changed. Let $\langle \beta_h, \widetilde{\beta}_h, \beta_v, \widetilde{\beta}_v \rangle$ be b-split of (Γ'_+, Γ'_-). Let $\overline{\Lambda_{\beta_h}}$ be a number of not avoided edges $(i, b) \in E$, i.e. $\overline{\Lambda_{\beta_h}} = \left|\mathcal{N}^- \cap \beta_h\right|$, and let $\overline{\Lambda_{\widetilde{\beta}_h}}$ be a number of not avoided edges $(b, i) \in E$, i.e $\overline{\Lambda_{\widetilde{\beta}_h}} = \left|\mathcal{N}^+ \cap \widetilde{\beta}_h\right|$. The (Γ'_+, Γ'_-) is a-feasible if $\overline{\Lambda_{\beta_h}} = 0$ and $\overline{\Lambda_{\widetilde{\beta}_h}} = 0$.*

5.2 SP neighborhood search schedule

Let (Γ_+, Γ_-) be a c- or a-feasible SP and let (Γ_+^D, Γ_-^D) be a (Γ_+, Γ_-) with removed job b. Let (Γ'_+, Γ'_-) be SP (Γ_+^D, Γ_-^D) with arbitrarily inserted job b and let G'_h be a horizontal constraint graph imposed by (Γ'_+, Γ'_-).

The SP neighborhood search schedule is given as Algorithm 1. It enumerates all (Γ'_+, Γ'_-), starting with $(\Gamma'_+, \Gamma'_-) = (\langle b \rangle \Gamma_+^D, \langle b \rangle \Gamma_-^D)$, i.e. $\langle b \rangle$ is concatenated with sequences Γ_+^D and Γ_-^D, respectively.

Denote by $M(j, \Gamma)$ an adjacent swap move that exchanges the element j with its right neighbor in sequence Γ.

Algorithm 1 Search schedule

```
1 : Set (Γ'+, Γ'−) = (⟨b⟩Γ+^D, ⟨b⟩Γ−^D)
2 : Evaluate(Γ'+, Γ'−)
3 : FOR i = 1 TO n DO:
4 :     FOR j = 1 TO n − 1 DO:
5 :         M(b, Γ'−)
6 :         Evaluate(Γ'+, Γ'−)
7 :     ENDFOR
8 :     IF i < n THEN
9 :         Set Γ'− = ⟨b⟩Γ−^D
10:         M(b, Γ'+)
11:         Evaluate(Γ'+, Γ'−)
12:     ENDIF
13: ENDFOR
```

In the inner-loop, lines 4 - 7, for a fixed position of b in Γ'_+, all possible positions of insertion of b into Γ'_- are successively evaluated by performing a series of adjacent swap moves of job b in Γ'_-. Next, in line 9 of the outer-loop, the changes made in the inner-loop to Γ'_- are canceled, and, in line 10, position of b in Γ'_+ is changed by an adjacent swap move. The `Evaluate`(Γ'_+, Γ'_-) evaluates feasibility and criterion of (Γ'_+, Γ'_-).

Let us analyze results of changes to (Γ'_+, Γ'_-) in the course of Algorithm 1. Let $\langle \beta_h, \widetilde{\beta}_h, \beta_v, \widetilde{\beta}_v \rangle$ be (b)-split of (Γ'_+, Γ'_-).

Since initially, according to (1), for each $i \in J \setminus \{b\}$ there exists horizontal relation $b \xrightarrow{h} i$ in $(\Gamma'_+, \Gamma'_-) = (\langle b \rangle \Gamma_+^D, \langle b \rangle \Gamma_-^D)$, then none of the edges (i, b), $i \in \mathcal{N}^-$ is contained in G'_h, whereas all edges (b, i), $i \in \mathcal{N}^+$ are contained. Therefore, in the c-feasibility case, initially $\Lambda_{\beta_h} = |\mathcal{N}^-|$ and $\Lambda_{\widetilde{\beta}_h} = 0$. Similarly, in the a-feasibility case, $\overline{\Lambda_{\beta_h}} = 0$ and $\overline{\Lambda_{\widetilde{\beta}_h}} = |\mathcal{N}^+|$. Let $\Lambda_{\widetilde{\beta}_v} = |\{x \in \widetilde{\beta}_v : x \in \mathcal{N}^-\}|$ (initially, $\Lambda_{\widetilde{\beta}_v} = 0$).

In the course of Algorithm 1, there are three cases of M operation, i.e. swapping of element b with its right neighbor either in Γ'_- or Γ'_+:

(i) Assume that before the swap move in line 5 of Algorithm 1 the SP is given as $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, b, \ldots, i, \ldots \rangle, \langle \ldots, b, i, \ldots \rangle)$. Then, after the adjacent swap move of b in Γ'_-, $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, b, \ldots, \underline{i}, \ldots \rangle, \langle \ldots, i, b, \ldots \rangle)$. According to Definition 9, this change to the SP implies $\widetilde{\beta}_h = \widetilde{\beta}_h \setminus \{i\}$, $\beta_v = \beta_v \cup \{i\}$. Observe, according to (1) and (2), a horizontal relation $b \xrightarrow{h} i$ is replaced by a vertical one $i \xrightarrow{v} b$. Therefore, if (b, i) is a requested edge ($(b, i) \in E$) then, after the move, $\Lambda_{\widetilde{\beta}_h} = \Lambda_{\widetilde{\beta}_h} + 1$. Similarly, if (b, i) has to be avoided, then $\overline{\Lambda_{\widetilde{\beta}_h}} = \overline{\Lambda_{\widetilde{\beta}_h}} - 1$.

(ii) Assume that before the swap move (line 5) $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, i, \ldots, b, \ldots \rangle, \langle \ldots, b, i, \ldots \rangle)$. Then, after the adjacent swap move of b in Γ'_-, $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, i, \ldots, b, \ldots \rangle, \langle \ldots, i, b, \ldots \rangle)$ and $\widetilde{\beta}_v = \widetilde{\beta}_v \setminus \{i\}$, $\beta_h = \beta_h \cup \{i\}$. Observe a vertical relation $b \xrightarrow{v} i$ is replaced by a horizontal one $i \xrightarrow{h} b$. Therefore, if $(i, b) \in E$ then, after the move, $\Lambda_{\beta_h} = \Lambda_{\beta_h} - 1$ and $\Lambda_{\widetilde{\beta}_v} = \Lambda_{\widetilde{\beta}_v} - 1$. Similarly, if $(i, b) \in E$ then, $\overline{\Lambda_{\beta_h}} = \overline{\Lambda_{\beta_h}} + 1$.

(iii) Assume that before the swap move (line 10) $(\Gamma'_+, \Gamma'_-) = (\langle \ldots, b, i, \ldots \rangle, \langle b, \ldots, i, \ldots \rangle)$. Then, after the adjacent swap move of b in Γ'_+, $(\Gamma'_+, \Gamma'_-) =$

$(\langle \ldots, i, b, \ldots \rangle, \langle b, \ldots, i, \ldots \rangle)$ and $\widetilde{\beta}_h = \widetilde{\beta}_h \setminus \{i\}$, $\widetilde{\beta}_v = \widetilde{\beta}_v \cup \{i\}$. Observe a horizontal relation $b \overset{h}{\rightarrow} i$ is replaced by a vertical one $b \overset{v}{\rightarrow} i$. Therefore, if $(b, i) \in E$ then, after the move, $\Lambda_{\beta_h} = \Lambda_{\beta_h} + 1$, and if $(i, b) \in E$ then $\Lambda_{\widetilde{\beta}_v} = \Lambda_{\widetilde{\beta}_v} + 1$. Similarly, if $(b, i) \in E$ then, $\overline{\Lambda_{\widetilde{\beta}_h}} = \overline{\Lambda_{\widetilde{\beta}_h}} - 1$.

5.3 The SP neighborhood evaluation algorithm

The Precel algorithm follows the search-schedule of Algorithm 1, but instead of managing (Γ'_+, Γ'_-) explicitly, it only maintains Λ- and $\overline{\Lambda}$-values and performs c- and a-feasibility evaluation with use of Lemmas 5 and 6, respectively. As Algorithm 2 we present an implementation of c-feasibility variant of the Precel algorithm for neighborhood evaluation of explicit-feasible solution-space, i.e., edges of G_p have to be contained in G'_h. Implementations for forced-feasible and possibly-feasible solution-spaces are analogous.

In order to identify the case of adjacent swap move in Γ'_-, the array $Match[j]$, $j \in \{1, \ldots, n\}$ is used, i.e., $Match[j] = k$ if $\Gamma_+[k] = j$ (we assume the k-th element of Γ is accessible via $\Gamma[k]$ operator). The `Push`(Λ) (`Pop`(Λ)) operation stores (restores) the state of Λ value on the heap before (after) inner-loop of the algorithm, respectively.

In subroutine `Eval`(b, i, j) a feasibility of an examined solution (insertion of b at positions i in Γ^D_+ and j in Γ^D_-) is evaluated with use of Lemma 5. Using the techniques from Janiak et al. (2010); Imahori et al. (2005), the makespan and the number of processors used in the schedule can be computed in the amortized constant time per examined solution.

The best found solution is recorded by storing its describing factors f^* (containing eg. workspan, feasibility mark, etc.) and corresponding insertion positions i^* and j^*. Construction of the best found solution can be done in $O(1)$ time after evaluation of the whole neighborhood.

Theorem 3 *The Precel algorithm evaluates feasibility of each solution in the insert-neighborhood of SP in an amortized $O(1)$ time per solution.*

Proof The initialization part (lines 1–4) takes $O(n)$. The `Eval` subroutine takes $O(1)$ for evaluating the condition stated in Lemma 5. The inner-loop (lines 8–18) takes $O(n)$. The `Push` and `Pop` operations take $O(1)$. A single pass of the outer-loop (lines 7–26) takes $O(n)$, what, with n passes, gives $O(n^2)$ in total and bounds the computational complexity of the overall algorithm. Since there is $O(n^2)$ solutions in the neighborhood, we get amortized $O(1)$ per single solution. □

5.4 Neighborhood pruning techniques

The run-time of the Precel algorithm can be additionally enhanced with the following properties, pruning the neighborhood during the search process.

Property 2 If $\Lambda_{\widetilde{\beta}_h} > 0$ after the move of case (i) in the Precel algorithm, then all further solutions constructed in the inner-loop are not feasible.

Algorithm 2 $PRECEL(\Gamma_+^D, \Gamma_-^D, \{b\}, G_p)$

```
1 : Initialize Match array
2 : Λ_{β_h} = deg⁻(b)
3 : Λ_{β̃_h} = 0
4 : Λ_{β̃_v} = 0
5 : Push(Λ_{β_h}, Λ_{β̃_h}, Λ_{β̃_v})
6 : (i*, j*, f*) = Eval(b, 0, 0)
7 : FOR i = 1 TO n DO:
8 :   FOR j = 1 TO n − 1 DO:
9 :     IF i ≤ Match[Γ_−^D[j]] THEN
10:       IF (b, Γ_−^D[j]) ∈ E_p THEN Λ_{β̃_h} += 1 ENDIF
11:     ELSE
12:       IF (Γ_−^D[j], b) ∈ E_p THEN
13:         Λ_{β_h} = Λ_{β_h} − 1
14:         Λ_{β̃_v} = Λ_{β̃_v} − 1
15:       ENDIF
16:     ENDIF
17:     (i*, j*, f*) = Eval(b, i − 1, j)
18:   ENDFOR
19:   IF i < n THEN
20:     Pop(Λ_{β_h}, Λ_{β̃_h}, Λ_{β̃_v})
21:     IF (Γ_+^D[i], b) ∈ E_p THEN Λ_{β̃_v} += 1 ENDIF
22:     IF (b, Γ_+^D[i]) ∈ E_p THEN Λ_{β_h} += 1 ENDIF
23:     (i*, j*, f*) = Eval(b, i, 0)
24:     Push(Λ_{β_h}, Λ_{β̃_h}, Λ_{β̃_v})
25:   ENDIF
26: ENDFOR
27: Insert b into (Γ_+^D, Γ_−^D) at (i*, j*)
```

Proof Assume after the move of case i) in the Precel algorithm $\Lambda_{\widetilde{\beta}_h} > 0$. The only operation that decreases the value of $\Lambda_{\widetilde{\beta}_h}$ is the `Pop All`(Λ) in the outer-loop of the Precel algorithm. Therefore, for each solution further examined in the inner-loop, $\Lambda_{\widetilde{\beta}_h} > 0$ and, according to Lemma 2, none of such solutions is feasible. □

Property 3 Consider the of case (i) in the Precel algorithm. If $(i, b) \in E_p$, then all further solutions constructed in the inner-loop are not feasible.

Proof The move of case (i) results in $\widetilde{\beta}_h = \widetilde{\beta}_h \setminus \{i\}$ and $\beta_v = \beta_v \cup \{i\}$. Until end of the inner-loop of the Precel algorithm, the job $i \in \beta_v$, and therefore, if $(i, b) \in E_p$ then no further solution in the inner-loop can satisfy this precedence constraint. □

Property 4 If $\Lambda_{\beta_h} > \Lambda_{\widetilde{\beta}_v}$ before the inner-loop in the Precel algorithm, then all solutions constructed in the inner-loop are not feasible.

Proof Assume $\Lambda_{\beta_h} > \Lambda_{\widetilde{\beta}_v}$ before the inner-loop. The only operation that increases the value of $\Lambda_{\widetilde{\beta}_v}$ is the move of the case (iii) in the outer-loop of the Precel algorithm. Therefore even if in the inner-loop are performed moves only of the case ii), $\Lambda_{\beta_h} - \Lambda_{\widetilde{\beta}_v} > 0$ and, according to Lemma 2, all solutions constructed in the inner-loop are not feasible. □

Property 5 If $\Lambda_{\widetilde{\beta}_h} > 0$ after the move of case (iii) in the Precel algorithm, then all further solutions from the neighborhood are not feasible.

Proof Assume after the move of case i) in the Precel algorithm $\Lambda_{\widetilde{\beta}_h} > 0$. There is no operation that decreases the value of $\Lambda_{\widetilde{\beta}_h}$ in the Precel algorithm once it becomes > 0 in the outer-loop. Therefore for each solution further examined in the neighborhood, $\Lambda_{\widetilde{\beta}_h} > 0$ and, according to Lemma 2, none of such solutions is feasible. □

The Algorithm 2 with applied Properties 2-5 is further referenced as the Precel+ algorithm. For the case of a-feasibility variant, the following property holds.

Property 6 If $\overline{\Lambda_{\beta_h}} > 0$ after the move of case (ii) in the Precel algorithm, then all further solutions constructed in the inner-loop are not feasible.

Proof Assume after the move of case (ii) in the Precel algorithm $\overline{\Lambda_{\beta_h}} > 0$. The only operation that decreases the value of $\overline{\Lambda_{\beta_h}}$ is the `Pop All`$(\overline{\Lambda})$ in the outer-loop of the Precel algorithm. Therefore, for each solution further examined in the inner-loop, $\overline{\Lambda_{\beta_h}} > 0$ and, according to Lemma 6, none of such solutions is feasible. □

6 Connectivity and diameter of the precedence constrained insert neighborhoods

In this section, we prove connectivity properties for insert neighborhoods of the explicit- and possible-feasibility solution spaces. Connectivity property guarantees that there exist a finite sequence of insert moves transforming any starting solution into any other (especially an optimal one) and is crucial for the efficiency of metaheuristics (Blum and Roli 2003). We first present a result for precedence constrained single permutations, and then we apply it to the SPs. Since presented proofs are constructive, we not only obtain bounds on the diameter of the considered solution spaces, but foremost the resulting algorithms can be independently applied as crossover or path-relinking operators in metaheuristics (see, e.g. Blum and Roli 2003).

6.1 A single permutation case

Let S be a space of all permutations of the set $B = \{1, \ldots, n\}$ and let G_p be reduced to its transitive reduction. Note that subtracted transitive edges are in fact redundant and are satisfied whenever a solution satisfies the reduced precedence graph.

Definition 10 Define an insert neighborhood of permutation $\pi \in S, \mathcal{N}(\pi, b)$, as a set of permutations obtained by performing all "insert" moves of the element b.

Definition 11 A neighborhood $\mathcal{N}(\pi) = \bigcup_{b \in B} \mathcal{N}(\pi, b)$ is called (OPT-) connected if, from each solution $\pi_S \in S$, any other (an optimal) solution $\pi_T \in S$ can be reached by a finite sequence $\pi_0 = \pi_S, \pi_1, \ldots, \pi_{k-1}, \pi_k = \pi_T$ of solutions $\pi_i \in S$ s.t. $\pi_{i+1} \in \mathcal{N}(\pi_i)$ for $i = 1, \ldots, k-1$.

Let $\pi_f = lcs(\pi_S, \pi_T)$ and $k = n - l(\pi_f)$, where $l(\pi_f)$ is the length of a sequence π_f and $lcs(\pi_S, \pi_T)$ is the longest common subsequence of π_S and π_T. From the works of Ulam (1972) and Fredman (1975) it follows that $\mathcal{N}(\pi)$ is a connected neighborhood (and therefore it is OPT-connected as well) and k is a minimal number of insert moves needed to transform π_S into π_T. Hunt and Szymanski (1977) showed an optimal algorithm computing $lcs(\pi_S, \pi_T)$ in $O(n \log \log n)$ time. Subsequently we show, that the same bounds apply to the precedence constrained case. Let S_P be a space of all explicit-feasible permutations of the set $B = \{1, \ldots, n\}$.

Definition 12 Define an insert neighborhood of permutation $\pi \in S_P$, $\mathcal{N}_P(\pi, b)$, as a set of feasible permutations obtained by performing all "insert" moves of the element b.

Definition 13 A neighborhood $\mathcal{N}_P(\pi) = \bigcup_{b \in B} \mathcal{N}_P(\pi, b)$ is called (OPT-) connected if, from each solution $\pi_S \in S_P$, any other (an optimal) solution $\pi_T \in S_P$ can be reached by a finite sequence $\pi_0, \pi_1, ..., \pi_{k-1}, \pi_k$ of solutions $\pi_i \in S_P$ s.t. $\pi_{i+1} \in \mathcal{N}_P(\pi_i)$ for $i = 1, \ldots, k-1$, $\pi_0 = \pi_S$ and $\pi_k = \pi_T$.

Subsequently, we show that $\mathcal{N}_P(\pi)$ is connected by giving an algorithm that computes a transformation between any two feasible permutations, consisting of minimal number of intermediate feasible permutations. Thus, we obtain a strict bound on the diameter of such a solution space.

Let $\pi_S, \pi_T \in S_P$ and let $\pi_f = lcs(\pi_S, \pi_T)$. The elements of $\pi_f = \langle f_1, \ldots, f_k \rangle$ are the *fixed-points* of transformation of π_S into π_T, i.e., elements of π_f are intact during the transformation (as in the unconstrained case).

Lemma 7 *Each element between consecutive fixed-points in π_S has to be moved either to the left of its closest left fixed-point, or to the right of its closest right fixed-point, in a transformation of π_S into π_T.*

Proof Let $\pi_f = lcs(\pi_S, \pi_T) = \langle f_1, \ldots, f_k \rangle$. Suppose the considered lemma is not true, and there is an element x such that $\langle \ldots f_i \ldots x \ldots f_{i+1} \ldots \rangle$ is a subsequence of both π_S and π_T. Then $\pi_f \neq lcs(\pi_S, \pi_T)$ what contradicts the assumption of the lemma. □

Denote by d_x a direction of a move of an element x. Let L_x denote closest left fixed point in π_S and R_x denotes closest right fixed point in π_S, for each element $x \in \pi_S \setminus \pi_f$. Let $pos(x)$, $pos(L_x)$ and $pos(R_x)$ be positions of x, L_x and R_x in π_T, respectively. If $pos(x) < pos(L_x)$ then $d_x =' \leftarrow'$, if $pos(x) > pos(R_x)$ then $d_x =' \rightarrow'$. For each $x \in \pi_f$ $d_x =' -'$.

Without precedence constraints elements of $B \setminus \pi_f$ are moved in an arbitrary order, whereas in the case of precedence constrained permutations, all intermediate solutions in a transformation of π_S into π_T have to be feasible permutations. Therefore, the sequence of moves must be performed in such an order, that preserves precedence constraints during the transformation.

First, note that the possibility of violating a precedence constraint $a \prec b$ is possible in two cases: when $d_a =' \rightarrow'$, $d_b =' \rightarrow'$ and the element a is moved before b (see, Fig. 4a), or when $d_a =' \leftarrow'$, $d_b =' \leftarrow'$ and the element b is moved before a. For the first case we say that b blocks a, for the latter, a blocks b. The algorithm PATH, given below, moves the elements in such an order that the above two cases are eliminated.

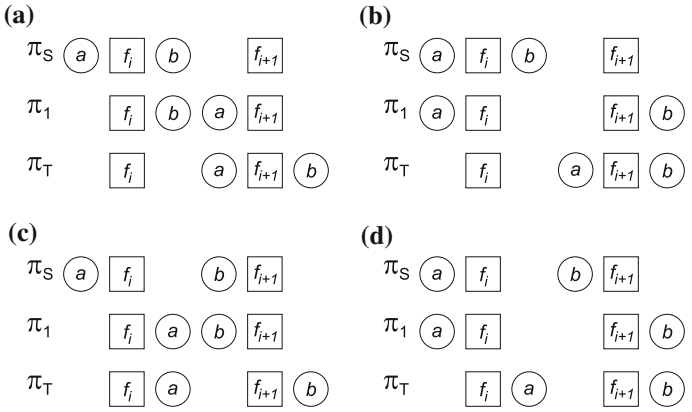

Fig. 4 Example transformations. Let $a \prec b$. **a** Constraint violation after move of a, **b** reversing the order prevents violation of constraint, **c** and **d** similar cases to **a** and **b** but without constraint violation

6.2 The PATH algorithm

Let us describe the PATH algorithm, presented as Algorithm 4. Given π_S and π_T, it first computes the fixed-points of the transformation, π_f, and determines a set $M = B \setminus \pi_f$ of elements to be moved. Next, for each element $j \in M$ it computes a direction of a move d_j and, on this basis it constructs a directed graph of blockades $G_B(M, E_B)$, where $E_B = \{(x, y) : x, y \in M, x \prec y, x \text{ blocks } y\}$.

Let π_i be an actual intermediate permutation of the transformation (initially equal to π_S). Moves in π_i are performed with use of reference points, i.e., elements that not going to change their relative positions in π_i in the further course of the transformation. Note that only fixed-points and already moved elements have fixed relative positions in π_i and can be reference points for performed moves (all other elements in π_i are going to change its positions). Reference points for moves performed on π_i are stored in EBT, van Emde Boas Tree (Emde et al. 1977) data structure for the *dynamic predecessor problem*. The following operations are performed on EBT:

- `INSERT`$(EBT, (x, p_x))$—adds an element x with the *search key* p_x, a position of x in π_T, to the set EBT;
- `PRED`(EBT, x)—predecessor query—returns an element of EBT with the largest search key that is less than a search key of an element x;
- $EBT \equiv$ `vEBTree`$(n + 1)$—constructs the data structure EBT to hold elements with search keys in the range $[0; n]$ and inserts an artificial element 0 with the key = 0.

Initially, all elements of π_f (fixed-points) are inserted to EBT.

The PATH algorithm iteratively processes elements to be moved according to the topological sorting of G_B. It picks x as the first element of M, removes x from M, and performs a move of x. First, the element x is inserted to EBT, as it reaches its destination position in the transformation. Next, a move of x is performed in π_i such that x is placed to the right of its predecessor in EBT.

To perform an arbitrary insert move in π_i in a constant time, consider π_i implemented as data structure $\mathcal{D}$, storing a permutation of a set $U = \{1, \ldots, n\}$ under an arbitrary sequence of move operations. The data structure $\mathcal{D}$ consist of a table $\mathcal{D}_T$ of $u + 1$ *nodes*. Each table entry (node) $\mathcal{D}_T[j]$ corresponds to an element $j \in U$ (each node stores its index). Nodes of $\mathcal{D}_T$ are linked to form a doubly-linked list $\mathcal{D}_L$ with $\mathcal{D}_T[0]$ as a *head*. An order of nodes in the list $\mathcal{D}_L$ represents a permutation of the set U. Observe that the design of $\mathcal{D}$ enables to locate any node $x \in U$ in the list $\mathcal{D}_L$ in the constant time.

Operations performed on $X \equiv \mathcal{D}$:

- $X \equiv \mathcal{D}(n)$—constructs an empty data structure X of type $\mathcal{D}$ with fixed universe $U = \{1, \ldots, n\}$. Initially the list $\mathcal{D}_L$ contains only head node. Construction of X takes $O(n)$ time and space.
- INIT$(X, \{x_1, \ldots, x_k\})$—initializes the data structure with given permutation $\langle x_1, \ldots, x_k \rangle$, $x_i \in U$. This operation takes $O(n)$ time.
- MOVE(X, j, k)—first removes $\mathcal{D}_T[j]$ from $\mathcal{D}_L$, and then inserts it back into $\mathcal{D}_L$ to the right to the node $\mathcal{D}_T[k]$. This operation takes $O(1)$ time.

The computational complexity of the PATH algorithm is $O(n \log \log n + |E_P|)$.

Algorithm 3 $PATH(\pi_S, \pi_T, G_P)$

```
1 : π_f =LCS(π_S, π_T)
3 : M = B \ π_f
4 : ∀_{j∈M} compute d_j
5 : construct G_B(M, E_B)
6 : order M according to the topological sorting of G_B
7 : π_i ≡ D(n);
8 : INIT(π_i, π_S)
9 : EBT ≡ vEBTree(n + 1);
10: ∀_{j∈π_f} INSERT(EBT, (j, pos(j)));
11: WHILE (M) ≠ ∅ DO:
12:    x = FIRST(M);
13:    DELETE(M, x);
14:    INSERT(EBT, (x, pos(x)));
15:    MOVE(π_i, x,PRED(EBT, x));
16: ENDWHILE
```

Theorem 4 *The algorithm PATH determines an optimal transformation of* $\pi_S \in S_P$ *into* $\pi_T \in S_P$, *i.e. a sequence* $\pi_0 = \pi_S, \pi_1, \ldots, \pi_{k-1}, \pi_k = \pi_T$ *of* $k = n - l(lcs(\pi_S, \pi_T))$ *solutions* $\pi_i \in S_P$ *s.t.* $\pi_{i+1} \in \mathcal{N}_P(\pi_i)$ *for* $i = 1, \ldots, k - 1$.

Proof First, we prove that there exist a topological ordering of G_B. Consider, there exist a path from a vertex u to v in G_B. According to a definition of G_B, vertices on that path have the same direction. There cannot exist a path from v to u since neither u or v cannot have the opposite direction. Since G_B is acyclic directed graph, it has a topological ordering.

Let x be processed element and let π_i be an actual intermediate permutation of the transformation of $\pi_S \in S_P$ into $\pi_T \in S_P$ and assume $\pi_i \in S_P$.

If a move of x does not violate any precedence constraint with its adjacent veritices in G_P, then it does not violate any transitive precedence constraint as well. Therefore, this is sufficient to prove that precedence constraints $z \prec x$ and $x \prec y$, where $z \in \mathcal{N}^-(x)$ and $y \in \mathcal{N}^+(x)$, are satisfied after the move of x. There are 12 cases depending on the directions of x, y and z.

It is easy to see that a move of x cannot violate any constraint adjacent with a vertex of direction $'-'$ or opposite to d_x; otherwise it would contradict feasibility of π_A or π_T. On the other hand, moving elements according to the topological ordering of G_B eliminates possible blockades and guarantees that the move cannot violate precedence constraints with adjacent vertices of the same direction as d_x.

After the move of x $\pi_i \in S_P$ and, by construction, x is placed in π_i such that elements of the set $B \setminus M$ form a common subsequence of π_i and π_T. Applying above steps iteratively we get $\pi_i = \pi_T$.

Each move of x in π_i is a move that belongs to $\mathcal{N}_P(\pi_i, x)$, and therefore the PATH produces a sequence $\pi_0, \pi_1, ..., \pi_{k-1}, \pi_k$ of k solutions $\pi_i \in S_P$ s.t. $\pi_{i+1} \in \mathcal{N}_P(\pi_i)$ for $i = 1, \ldots, k-1$, $\pi_0 = \pi_S$ and $\pi_k = \pi_T$. Since the PATH algorithm performs moves of the same elements as in the unconstrained case, the $k = n - l(lcs(\pi_S, \pi_T))$ is a minimal number of moves. □

Corollary 3 *The neighborhood $\mathcal{N}_P(\pi_A)$ is connected and OPT-connected.*

Corollary 4 *The diameter of the solution space under $\mathcal{N}_P$ equals $n - 1$.*

6.3 Connectivity of precedence constrained sequence pair neighborhoods

In this section we show connectivity properties of considered SP neighborhoods. Let S^2 be a space of all SPs of the set $J = \{1, \ldots, n\}$, let S^2_P be S^2 limited to possibly-feasible SPs and let S^2_E be S^2 limited to explicit-feasible SPs. Subsequently, we show that S^2_P and S^2_E are connected under the insert neighborhood.

6.3.1 Explicit-feasibility case

Definition 14 Define an insert neighborhood of an explicit-feasible SP $(\Gamma_+, \Gamma_-) \in S^2_E$, $\mathcal{N}_E(\Gamma_+, \Gamma_-, b)$, as a set of explicit-feasible SPs obtained by performing all possible "insert" moves of the element b.

Definition 15 A neighborhood $\mathcal{N}_E(\Gamma_+, \Gamma_-) = \bigcup_{b \in B} \mathcal{N}_E(\Gamma_+, \Gamma_-, b)$ is called (OPT-)connected if, from each solution $(\Gamma^S_+, \Gamma^S_-) \in S^2_E$, any other (an optimal) solution $(\Gamma^T_+, \Gamma^T_-) \in S^2_E$ can be reached by a finite sequence (Γ^0_+, Γ^0_-), (Γ^1_+, Γ^1_-), $\ldots$, $(\Gamma^{k-1}_+, \Gamma^{k-1}_-)$, (Γ^k_+, Γ^k_-) of solutions $(\Gamma^i_+, \Gamma^i_-) \in S^2_E$ such that $(\Gamma^{i+1}_+, \Gamma^{i+1}_-) \in \mathcal{N}_E(\Gamma^i_+, \Gamma^i_-)$ for $i = 1, \ldots, k-1$, $(\Gamma^0_+, \Gamma^0_-) = (\Gamma^S_+, \Gamma^S_-)$ and $(\Gamma^k_+, \Gamma^k_-) = (\Gamma^T_+, \Gamma^T_-)$.

Theorem 5 *The precedence constrained insert neighborhood of SP, $\mathcal{N}_E(\Gamma_+, \Gamma_-)$, $(\Gamma_+, \Gamma_-) \in S^2_E$, is connected and OPT-connected.*

Proof According to Definition 15, $\mathcal{N}_E(\Gamma_+, \Gamma_-)$ is connected if there exist a finite sequence of moves in $\mathcal{N}_E(\Gamma_+, \Gamma_-)$ that transform any $(\Gamma^S_+, \Gamma^S_-) \in S^2_E$ into

$(\Gamma_+^T, \Gamma_-^T) \in S_E^2$. According to Theorem 4, there exist finite sequences of moves transforming Γ_+^S into Γ_+^T and Γ_-^S into Γ_-^T such that each intermediate permutation is explicit-feasible.

Based on a methodology from Wakata et al. 2003, consider a transformation $(\Gamma_+^S, \Gamma_-^S), \ldots, (\Gamma_+^T, \Gamma_-^S), \ldots, (\Gamma_+^T, \Gamma_-^T)$. Since each single permutation appearing in the transformation is explicit-feasible, according to Corollary 2 and Theorem 2, each intermediate SP is explicit-feasible as well. The $\mathcal{N}_E(\Gamma_+, \Gamma_-)$ is connected and therefore it is OPT-connected. □

Note, using insert moves in the transformation, in Theorem 5 we get better bound than can be obtained by bubble-sort based-methodology from Wakata et al. (2003).

6.3.2 Possibly-feasibility case

Definition 16 Define an insert neighborhood of a possibly-feasible SP $(\Gamma_+, \Gamma_-) \in S_P^2$, $\mathcal{N}_P(\Gamma_+, \Gamma_-, b)$, as a set of possibly-feasible SPs obtained by performing all possible "insert" moves of the element b.

Definition 17 A neighborhood $\mathcal{N}_P(\Gamma_+, \Gamma_-) = \bigcup_{b \in B} \mathcal{N}_P(\Gamma_+, \Gamma_-, b)$ is called (OPT-)connected if, from each solution $(\Gamma_+^S, \Gamma_-^S) \in S_P^2$, any other (an optimal) solution $(\Gamma_+^T, \Gamma_-^T) \in S_P^2$ can be reached by a finite sequence $(\Gamma_+^0, \Gamma_-^0), (\Gamma_+^1, \Gamma_-^1), \ldots, (\Gamma_+^{k-1}, \Gamma_-^{k-1}), (\Gamma_+^k, \Gamma_-^k)$ of solutions $(\Gamma_+^i, \Gamma_-^i) \in S_P^2$ such that $(\Gamma_+^{i+1}, \Gamma_-^{i+1}) \in \mathcal{N}_P(\Gamma_+^i, \Gamma_-^i)$ for $i = 1, \ldots, k-1$, $(\Gamma_+^0, \Gamma_-^0) = (\Gamma_+^S, \Gamma_-^S)$ and $(\Gamma_+^k, \Gamma_-^k) = (\Gamma_+^T, \Gamma_-^T)$.

Theorem 6 *The precedence constrained insert neighborhood of SP, $\mathcal{N}_P(\Gamma_+, \Gamma_-)$, $(\Gamma_+, \Gamma_-) \in S_P^2$, is connected and OPT-connected.*

Proof Consider a transformation $(\Gamma_+^S, \Gamma_-^S), \ldots, (\Gamma_+^S, \Gamma_0), \ldots, (\Gamma_+^T, \Gamma_0), \ldots, (\Gamma_+^T, \Gamma_-^T)$, where Γ_0 is a topological ordering of vertices of precedence constraint graph G_p. Subsequently we construct such a transformation and show, that each intermediate SP is possibly-feasible.

Let Γ_A, Γ_B and Γ_C are permutations of the set J, and (Γ_A, Γ_B) and (Γ_A, Γ_C) are possibly-feasible SPs. Let (Γ_A, Γ) be an intermediate SP of transformation between (Γ_A, Γ_B) and (Γ_A, Γ_C). According to Definition 6 and Property 1, (Γ_A, Γ) is not possibly-feasible only if $(\Gamma_A, \Gamma) = (\langle \ldots, b, \ldots, a, \ldots \rangle, \langle \ldots, b, \ldots, a, \ldots \rangle)$ for some $(a, b) \in G_p$. Therefore, in order to (Γ_A, Γ) be possibly-feasible, if $\Gamma_A = \langle \ldots, b, \ldots, a, \ldots \rangle$ then $\Gamma = \langle \ldots, a, \ldots, b, \ldots \rangle$, i.e., a must precede b in Γ. Let $G_p^A(j, E_p^A)$ be a precedence constraint graph enforced by Γ_A, where $(a, b) \in E_p^A$ if $\Gamma_A = \langle \ldots, b, \ldots, a, \ldots \rangle$ and $(a, b) \in TC(G_p)$, and $TC(G_p)$ is a transitive closure of G_p. According to Theorem 4, there exist a transformation between permutations Γ_B and Γ_C such that G_p^A is satisfied by each intermediate permutation. Therefore, there exists a transformation between (Γ_A, Γ_B) and (Γ_A, Γ_C) s.t. each SP is possibly-feasible.

Sequence pairs (Γ_+^S, Γ_-^S) and (Γ_+^T, Γ_-^T) are possibly-feasible by assumption, (Γ_+^T, Γ_0) and (Γ_+^S, Γ_0) are also possibly-feasible, since Γ_0 satisfies all precedence constraints. Therefore, $\mathcal{N}_P(\Gamma_+, \Gamma_-)$ is connected and OPT-connected. □

Corollary 5 *If* (Γ_+^S, Γ_-^T) *is possibly-feasible, then a transformation of Theorem* 6 *can be shortened to* $(\Gamma_+^S, \Gamma_-^S), \ldots, (\Gamma_+^S, \Gamma_-^T), \ldots, (\Gamma_+^T, \Gamma_-^T)$. *Similarly, if* (Γ_-^S, Γ_+^T) *is possibly-feasible, then a transformation of Theorem* 6 *can be shortened to* $(\Gamma_+^S, \Gamma_-^S), \ldots, (\Gamma_-^S, \Gamma_+^T), \ldots, (\Gamma_+^T, \Gamma_-^T)$.

7 Experimental evaluation of sequence pair interpretations

In this section, we investigate the impact of SP interpretations and corresponding neighborhood structures on the efficiency of algorithms solving the considered scheduling problem. To not test the proposed approach on just random solutions, the experiment was designed to utilize most common approaches to problem solving—we developed constructive heuristic based on NEH (Nawaz et al. 1983) as a generator of starting solutions and metaheuristics based on the Simulated Annealing (SA) framework. Both considered techniques has been successively applied in the vast number of areas, e.g., Janiak et al. (2010); Blum and Roli (2003); Blum et al. (2008).

7.1 Constructive heuristic

The presented approximate constructive algorithm resembles the *insertion technique* of NEH (Nawaz et al. 1983), primarily applied to the flow-shop scheduling, where it is still among best constructive heuristics in that field.

Let $O = (o_1, \ldots, o_n)$ be some order of jobs $J = \{1, \ldots, n\}$. The solution is constructed by successively inserting jobs $o_1, \ldots, o_n$ into an initially empty Sequence Pair. In each insertion step, all possible insertion positions of job o_i into $(\Gamma_+^{i-1}, \Gamma_-^{i-1})$ are evaluated. The algorithm first picks all explicit-feasible solutions from the neighborhood using the Precel algorithm, and from them the one that minimizes workspan criterion. The best solution from the neighborhood creates a base for the next step. The pseudo-code of the *PREcedence constrained - Greedy Insertion Technique* (PRE-GIT) algorithm is given as Algorithm 4. The time complexity of the PRE-GIT algorithm is only $O(n^3)$ due to efficiency of the Precel algorithm.

Algorithm 4 PRE-GIT(J, G_p)

```
1: Define order O
2: FOR i = 1 TO n DO:
3:     G_p^i = G_p[{o_1, ..., o_i}]
4:     (Γ_+^i, Γ_-^i) = PRECEL(Γ_+^{i-1}, Γ_-^{i-1}, o_i, G_p^i)
5: ENDFOR
6: RETURN: (Γ_+^n, Γ_-^n)
```

Observe that at each step the algorithm deals with an incomplete solution, so the feasibility and the workspan are computed only for a constructed so far fragment of a solution. Therefore, in the Precel neighborhood evaluation algorithm we consider

for each iteration i precedence relations between jobs are given by $G_p^i(V^i, E_p^i)$, a subgraph of G_p induced by $V^i = \{o_1, \ldots, o_i\}$, i.e. $E_p^i = \{(u, v) \in E_p : u, v \in V^i\}$.

7.2 Simulated annealing

The search process of SA is a semi-random trajectory in the search space, biased towards the "good" region of solution space. In its basic form SA combines an intensification strategy (an iterative local search using a neighborhood concept) with a form of diversification (by so-called cooling-schedule allowing uphill moves to escape from the local minimum). Its efficiency, i.e. the criterion value of delivered solution, strongly depends on the number of visited solutions in the search space. This is especially important in the case when search space is filled with infeasible solutions, where SA performs a lot of futile iterations.

The obvious improvement of this strategy is to limit the neighborhood to contain only feasible solutions. However, while it may increase the effectiveness of the search process, it may be its killer as well, if the restricted neighborhood does not have an OPT-connectivity property. The guarantee that from any starting solution in the search space, an optimal solution can be reached by a finite sequence of moves (the OPT-connectivity property of neighborhood) is the most important condition allowing SA to converge (with probability) to the global optimum.

The SA starts the search process from a feasible SP, delivered by a constructive heuristic. Then, at each iteration SA performs a random deletion of a job (say, b) from the actual solution S followed by the evaluation of considered SP neighborhood, i.e. all possible insertion positions of b are evaluated, and the new solution S' is randomly sampled from this neighborhood. Let the workspan $W_{max}(S)$ be the objective function to be minimized. In the case the new solution S' is feasible, it replaces the old solution S either with a probability computed following the Boltzmann distribution $exp(-\frac{W_{max}(S')-W_{max}(S)}{\tau})$ if $W_{max}(S') \geq W_{max}(S)$, or without a draw if $W_{max}(S') < W_{max}(S)$. The temperature τ is decreased after each iteration $iter = 1, 2, \ldots, MaxIter$ by a geometric cooling schedule, i.e., $\tau_{iter} = \alpha\tau_{iter-1}$, $\alpha \in (0, 1)$. We set $\tau_0 = 1000, \alpha = 0.998$ and $MaxIter = 5000$ verifying that in each case the annealing stabilizes with a good solution, i.e., the best recorded solution has not been improved by many iterations.

7.3 Numerical experiment

To evaluate the presented SP interpretations, we performed a numerical experiment, in which we compared run-times and quality of delivered solutions of SA algorithms. The algorithms were coded in C++ and simulations were run on Intel Core i7 3610QM CPU under Windows 7 Operating System. We consider SA_A to be the SA operating in the forced-feasibility solution space, SA_B to be the SA operating in the explicit-feasibility solution space, SA_C to be the SA operating in the possible-feasibility solution space, and SA_D to be the SA operating in the plain SP solution space. In the case of SA_A, SA_B and SA_C, the random move in SA is performed on a neighborhood containing only

Table 1 Results of SA started from PRE-GIT-A

n	$\|E_P\|$	σ_{SA_A}	σ_{SA_B}	σ_{SA_C}	σ_{SA_D}	t_{SA_A}	t_{SA_B}	t_{SA_C}	t_{SA_D}
50	50	−13.17	−16.36	−12.09	−11.88	0.482	0.108	0.212	0.064
	85	−11.83	−17.46	−10.99	−9.70	0.502	0.087	0.211	0.069
	250	−3.08	−11.55	−5.24	−3.52	0.493	0.047	0.203	0.072
	550	−1.89	−10.12	−3.95	−2.40	0.574	0.043	0.236	0.092
	1000	−1.11	−6.99	−2.83	−1.32	0.552	0.039	0.226	0.093
100	100	−16.15	−18.99	−14.39	−14.38	2.446	0.486	0.999	0.263
	200	−6.19	−13.21	−5.39	−5.16	2.268	0.291	0.893	0.266
	600	−1.45	−9.89	−1.64	−0.94	2.289	0.168	0.931	0.337
	1000	−1.17	−12.73	−2.08	−1.16	2.009	0.115	0.816	0.306
	2000	−0.59	−9.17	−1.10	−0.56	1.947	0.106	0.810	0.328
200	200	−11.76	−15.77	−10.15	−9.81	8.948	1.718	3.621	0.880
	460	−5.56	−11.78	−4.03	−3.69	9.367	0.953	3.585	1.108
	1600	−0.93	−12.43	−1.23	−0.74	8.138	0.396	3.289	1.239
	2800	−0.80	−13.21	−0.75	−0.56	7.785	0.338	3.169	1.220
	5500	−0.33	−12.52	−0.54	−0.24	7.060	0.305	2.918	1.158
300	300	−5.10	−12.10	−3.40	−7.37	19.644	3.842	7.892	1.821
	740	−3.80	−11.64	−2.06	−2.56	19.964	1.859	7.743	2.323
	3000	−0.85	−13.29	−0.91	−0.38	17.140	0.735	7.303	2.750
	5200	−0.52	−14.00	−0.55	−0.23	16.215	0.620	6.713	2.543
	10,000	−0.26	−13.19	−0.37	−0.19	15.422	0.587	6.526	2.473

feasible solutions, filtered by an appropriate variant of Precel algorithm. The starting solutions were constructed by PRE-GIT heuristic. For the PRE-GIT-A algorithm jobs were ordered according to non-increasing workloads of jobs ($w_i \cdot h_i, i = 1, \ldots, n$). For the PRE-GIT-B jobs were ordered according to topological sorting of G_p. Starting SA from solutions delivered by constructive heuristic enables the test to focus on the question how efficient are particular SA algorithms (SP interpretations) in improving the initial solution.

The proposed algorithms were evaluated for the problem sizes of $n \in \{50, 100, 200, 300\}$. For each problem size n, we considered the constraint graphs of $|E_P| \in \{n, n \log n, \frac{n\sqrt{n}-n \log n}{2}, n\sqrt{n}, 2n\sqrt{n}\}$ edges (before augmentation to transitive closure), i.e., we tested a behavior of presented algorithms on a wide range of graph densities. For each pair of n and $|E_P|$, 100 different random instances were generated. Parameters of jobs were randomly sampled from the uniform distribution over the integers in the range: $w_i, h_i \in [1; 100]$. Precedence constraint graphs were generated in such a way, that for each edge $(a, b) \in G_p, b > a$.

For each SA algorithm $A \in \{SA_A, SA_B, SA_C, SA_D\}$ and heuristic $H \in \{PRE - GIT - A, PRE - GIT - B\}$, for each instance I, we measured a run-time $t_A(I)$ of algorithm A on instance I, relative improvements of the criterion value over the initial solution $\sigma_A(I) = \frac{W^A_{max}(I) - W^H_{max}(I)}{W^H_{max}(I)} \cdot 100\,\%$, where $W^A_{max}(I)$ and $W^H_{max}(I)$ denote

Table 2 Results of SA started from PRE-GIT-B

n	$\|E_P\|$	σ_{SA_A}	σ_{SA_B}	σ_{SA_C}	σ_{SA_D}	t_{SA_A}	t_{SA_B}	t_{SA_C}	t_{SA_D}
50	50	−19.09	−22.66	−17.95	−17.07	0.679	0.149	0.295	0.086
	85	−11.85	−17.82	−10.38	−9.30	0.655	0.108	0.270	0.082
	250	−4.76	−13.00	−4.82	−4.65	0.655	0.060	0.262	0.091
	550	−2.12	−9.80	−3.10	−2.37	0.523	0.039	0.215	0.083
	1000	−1.31	−9.23	−2.70	−1.71	0.495	0.033	0.196	0.080
100	100	−12.12	−15.08	−11.00	−10.59	2.573	0.514	1.050	0.278
	200	−10.00	−15.94	−7.72	−7.21	2.499	0.304	0.961	0.274
	600	−3.47	−12.79	−3.03	−2.53	2.357	0.148	0.921	0.308
	1000	−1.32	−9.11	−1.32	−0.84	2.115	0.117	0.860	0.310
	2000	−0.38	−7.99	−0.46	−0.34	1.958	0.102	0.802	0.312
200	200	−11.77	−15.25	−9.54	−9.93	9.243	1.748	3.708	0.880
	460	−4.80	−8.93	−3.04	−2.66	9.151	0.974	3.672	1.070
	1600	−1.95	−8.60	−1.32	−0.95	8.410	0.409	3.389	1.226
	2800	−0.94	−7.62	−0.67	−0.32	7.769	0.304	3.099	1.144
	5500	−0.28	−6.49	−0.37	−0.14	7.180	0.278	2.933	1.091
300	300	−4.96	−13.10	−2.64	−9.35	19.208	3.793	8.273	1.849
	740	−3.89	−9.85	−1.55	−3.23	19.457	1.894	8.115	2.332
	3000	−0.96	−5.52	−0.59	−0.33	17.858	0.674	7.248	2.649
	5200	−0.33	−4.31	−0.25	−0.12	16.239	0.573	6.773	2.412
	10,000	−0.35	−5.59	−0.22	−0.08	15.580	0.516	6.244	2.298

the criterion values provided by algorithms from A and H, respectively. The average values t_A (given in seconds) and σ_A collected in the experiment are presented in Table 1 (for H = PRE-GIT-A) and Table 2 (for H = PRE-GIT-B).

From the results reported in Tables 1 and 2 it can be seen that SA_B outperforms other competing algorithms both in terms of run-time and quality of delivered solutions, even thorough solution space with explicit-feasibility is the most limited interpretation of SP. Especially, for dense constraint graphs, SA_B considerably improved initial solutions (by 9 % for $n = 50$ to 5 % for $n = 300$ when started from PRE-GIT-B) while competitors was merely able to do so (by 2.7 % for $n = 50$ to 0.33 % for $n = 300$). Simultaneously, SA_B was few times faster on dense graphs, and slower only by SA_D on sparse constraint graphs. This shows the impact of neighborhood pruning properties on the run-time of the algorithm.

On the other hand, the performance of SA_A was comparable to SA_C and SA_D, even the solution space with forced-feasibility has the optimality of representation property. Although, a connectivity status of forced-feasibility insert neighborhood is an open question, this result suggests that it is either not connected, or connected in a limited degree, i.e., in a way that makes algorithm unable to escape from the bad region of solution space. Moreover, the schedule evaluation complexity and high hidden cost of neighborhood evaluation makes SA_A up to 30 times slower than SA_B. While SA_C

was 3 times slower than SA_D, it always gave better results than SA_D (thought not by much).

The experiment clearly shows the advantages of the proposed method of solving the precedence constrained scheduling problems, in which solutions can be represented by either explicit-feasible permutations (single processor cases) or SPs (multiprocessor task scheduling problems).

8 Conclusions

In this paper, we showed that SP representation, primarily applied to the rectangle packing problems appearing in the VLSI industry, can be an simple representation of precedence constraints. The SP can span the solution-space of suitable scheduling problems, or represent only a space of directed acyclic graphs, without additional interpretation of its vertices and edges. In that settings, we constructed an incremental precedence constrained SP neighborhood evaluation algorithm, computing feasibility of each solution in the insert neighborhood in an amortized constant time per examined solution, and proved the connectivity property of the considered neighborhood, by showing an algorithm computing Ulam's metric for the precedence constrained case of permutations. Since the length of the optimal number of insert moves in the unconstrained SP case can be viewed as a two-dimensional Ulam's metric, derivation of its properties, as well as properties of its precedence constrained counterpart will constitute the directions of our future research.

Acknowledgements The author is grateful to the Referees for their valuable comments and suggestions. This work was supported by The National Science Centre, Poland, under Project No. 2012/05/D/HS4/01129.

References

Adya S, Markov I (2003) Fixed-outline floorplanning: enabling hierarchical design. IEEE Trans Very Large Scale Integr Syst 11(6):1120–1135

Agnetis A, Kellerer H, Nicosia G, Pacifici A (2012) Parallel dedicated machines scheduling with chain precedence constraints. Eur J Oper Res 221(2):296–305

Augustine J, Banerjee S, Irani S (2009) Strip packing with precedence constraints and strip packing with release times. Theor Comput Sci 410(38–40):3792–3803

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112(1):3–41

Brucker P, Knust S (2012) Complex scheduling. Springer, Berlin

Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35(3):268–308

Blum C, Aguilera M, Roli A, Sampels M (2008) Hybrid metaheuristics. An emerging approach to optimization. Studies in computational intelligence, vol. 114

Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics, vol 57. Springer, New York, pp 457–474

Burke E, Gendreau M, Hyde M, Kendall G, Ochoa G, Ozcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64:1695–1724
Chandra C, Liu Z, He J, Ruohonen T (2014) A binary branch and bound algorithm to minimize maximum scheduling cost. Omega 42(1):9–15
Chen B, Potts CN, Woeginger GJ (1999) A review of machine scheduling: complexity, algorithms and approximability. In: Du D-Z, Pardalos PM (eds) Handbook of combinatorial optimization, vol 3. Springer, Berlin, pp 1493–1641
Chen C, Chu C (2012) A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. IEEE Trans Parallel Distrib Syst 24:1479–1488
Debczynski M, Gawiejnowicz S (2013) Scheduling jobs with mixed processing times, arbitrary precedence constraints and maximum cost criterion. Comput Ind Eng 64(1):273–279
Driessel R, Monch L (2011) Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. Comput Ind Eng 61(2):336–345
Drozdowski M (1996) Scheduling multiprocessor tasks—an overview. Eur J Oper Res 94:215–230
Duin CW, Sluis EV (2006) On the complexity of adjacent resource scheduling. J Sched 9(1):49–62
Fredman M (1975) On computing the length of longest increasing subsequences. Discret Math 11:29–35
Graham R, Lawler E, Lenstra J, Kan AR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Proceedings of the advanced research institute on discrete optimization and systems applications of the systems science panel of NATO and of the discrete optimization symposium, pp 287–326
Günther E, König FG, Megow N (2014) Scheduling and packing malleable and parallel tasks with precedence constraints of bounded width. J Comb Optim 27(1):164–181
Huang E, Korf RE (2013) Optimal rectangle packing: an absolute placement approach. J Artif Intell Res 46:47–87
Hunt J, Szymanski T (1977) A fast algorithm for computing longest common subsequences. Commun ACM 20(5):350–353
Imahori S, Yagiura M, Ibaraki T (2003) Local search algorithms for the rectangle packing problem with general spatial costs. Math Progr 97:543–569
Imahori S, Yagiura M, Ibaraki T (2005) Improved local search algorithms for the rectangle packing problem with general spatial costs. Eur J Oper Res 167:48–67
Janiak A, Kozik A, Lichtenstein M (2010) New perspectives in VLSI design automation: deterministic packing by sequence pair. Ann Oper Res 179:35–56
Jansen K, Thöle R (2010) An approximation algorithm for scheduling malleable tasks under general precedence constraints. SIAM J Comput 39(8):3571–3615
Jansen K, Zhang H (2012) Scheduling malleable tasks with precedence constraints. J Comput Syst Sci 78(1):245–259
Kolisch R, Padman R (2001) An integrated survey of deterministic project scheduling. Omega 29(3):249–272
Korf R (2003) Optimal rectangle packing: initial results. In: Proceedings of the thirteenth international conference on automated planning and scheduling (ICAPS 2003), Trento, Italy, 9–13 June 2003, pp 287–295
Kozik A (2013) Fully dynamic evaluation of sequence pair. IEEE Trans Comput-Aided Des Integr Circuits Syst 32(6):894–904
Lai J, Lin M-S, Wang T-C, Wang L-C (2001) Module placement with boundary constraints using the sequence-pair representation. In: Proceedings of the 2001 Asia and South Pacific design automation conference, pp 515–520
Li SC, Leong HW, Quek SK (2004) New approximation algorithms for some dynamic storage allocation problems. In: Chwa K-Y, Munro JI (eds) Computing and combinatorics, vol 3106., Lecture notes in computer scienceSpringer, Berlin, pp 339–348
Lim A, Ma H, Rodrigues B, Tan S, Xiao F (2013) New meta-heuristics for the resource-constrained project scheduling problem. Flex Serv Manuf J 25(1–2):48–73
Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: a survey. Eur J Oper Res 141(2):241–252
Mitola J, Maguire G (1999) Cognitive radio: making software radios more personal. IEEE Pers Commun Mag 6(4):13–18

Murata H, Fujiyoshi K, Nakatake S, Kajitani Y (1996) VLSI module placement based on rectangle-packing by the sequence pair. IEEE Trans CAD ICs 15:1518–1524

Nawaz M, Enscore E Jr, Ham I (1983) A heuristic algorithm for m-machine, n-job flow-shop sequencing problem. OMEGA 11:91–95

Peltz C (2003) Web services orchestration and choreography. IEEE Comput 36:46–52

Rudek R, Rudek A, Kozik A (2013) The solution algorithms for the multiprocessor scheduling with workspan criterion. Expert Syst Appl 40:2799–2806

Svensson O (2011) Hardness of precedence constrained scheduling on identical machines. SIAM J Comput 40(5):1258–1274

Świątek P, Stelmach P, Prusiewicz A, Juszczyszyn K (2012) Service composition in knowledge-based SOA systems. New Gener Comput 30(2–3):165–188

Takahashi T (1996) An algorithm for finding a maximum-weight decreasing sequence in a permutation. IEICE Tech Res Rep VLSI Des Technol 7:31–35

Tang X, Tian R, Wong D (2001) Fast evaluation of sequence pair in block placement by longest common subsequence computation. IEEE Trans CAD ICs 20(12):1406–1413

Turek J, Wolf JL, Yu PS (1992) Approximate algorithms for scheduling parallelizable tasks. In: Proceedings of the 4th annual ACM symposium on parallel algorithms and architectures (SPAA), pp 323–332

Ulam S (1972) Some ideas and prospects in biomathematics. Ann Rev Biophys Bioeng 1:277–292

van Emde Boas P, Kaas R, Zijlstra E (1977) Design and implementation of an efficient priority queue. Math Syst Theory 10:99–127

Wakata K, Saito H, Fujiyoshi K, Sakanushi K, Obata T, Kodama C (2003) An improved method of convex rectilinear block packing based on sequence-pair. IEICE Trans Fundam Electron Commun Comput Sci E86–A/12:3148–3157

Wong CS, Chan FTS, Chung SH (2013) A joint production scheduling approach considering multiple resources and preventive maintenance tasks. Int J Prod Res 51(3):883–896

Young EFY, Chu CCN, Ho ML (2004) Placement constraints in floorplan design. IEEE Trans Very Large Scale Integr Syst 12(7):735–745

Zhao B, Aydin H, Zhu D (2013) Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. ACM Trans Des Autom Electron Syst 18(2):23:1–23:21